IDL_Guide Documentation

Release 0.9

us

Dec 02, 2019

CONTENTS

This package can be used to fit an Implicit Deep Learning (IDL) model for regression and classification purpose.

CHAPTER

ONE

INTRODUCTION

This package can be used to fit an Implicit Deep Learning (IDL) model for regression and classification purpose.

The IDL.fit function estimates a vector of parameters by applying successively gradient descents (see more at *Gradient Descents*) and dual ascent (see more at *Dual Ascents*).

1.1 Implicit Deep Learning

** section 1.1 **

Given an input $u \in \mathbb{R}^n$, where n denotes the number of features, we define the implicit deep learning prediction rule $\hat{y}(u) \in \mathbb{R}^n$ with ReLU activation

$$\hat{y}(u) = ...$$

1.2 Loss Functions

** section 3.2 equation 4 ** + Classification loss (see more at *Learning Process*)

1.3 Description of the learning process

(see more at *Formulation for IDL*)

1.4 Description of the prediction process

(see more at *Predicting*)

1.5 Setup

TODO

The package is compatible with Python version 3 or higher only. The user is expected to have installed cvxpy before running the package. Go to … for more information.

1. Switch to a proper directory and then type:

```
git clone + https://github.com/...
```

CHAPTER TWO

FORMULATION FOR IDL

See *Citing* for in-depth explanation

2.1 Problem Formulation

** section 3.1 **

2.2 Fenchel Divergence Lagrangian Relaxation

** section 3.2 **

2.3 Linear matrix inequality

** section 3.3 **

2.4 Bi-convex Formulation

`utilities.GradientDescents.`**`gradient_descent_theta`**(*theta*, *X*, *U*, *Y*)

Returns the gradient of theta :param theta: a dictionary :param X: hidden variables :param U: input data :param Y: output data :return: grad_theta: dictionary containing gradients of elemnts in theta

CHAPTER THREE

LEARNING PROCESS

** section 3.4 algo in the end **

class IDL.**IDLModel**(*hidden_features=1*, *alpha=0.1*, *epsilon=0.1*, *random_state=0*, *seed=None*)

demo_param [str, default='demo_param'] A parameter used for demonstation of how to pass and store paramters.

fit(*X*, *y*, *verbose=1*, *rounds_number=100*, *sample_weight=None*, *eval_set=None*, *eval_metric=None*, *early_stopping_rounds=None*, *callbacks=None*)

Fit the IDLModel parameters

X [{array-like, sparse matrix}, shape (m_samples, n_features)] The training input samples.

y [array-like, shape (m_samples,) or (m_samples, p_outputs)] The target values (class labels in classification, real numbers in regression).

verbose: If True (1) then we print everything on the console rounds_number : how many rounds max do you want to do early_stopping : If True : once the L2Loss will increase we automatically stop sample_weight : array_like

instance weights

eval_set [list, optional] A list of (X, y) tuple pairs to use as a validation set for early-stopping

eval_metric [str, callable, optional] If a str, should be a built-in evaluation metric to use. See doc/parameter.rst. If callable, a custom evaluation metric. The call signature is func(y_predicted, y_true) where y_true will be a DMatrix object such that you may need to call the get_label method. It must return a str, value pair where the str is a name for the evaluation and value is the value of the evaluation function. This objective is always minimized.

early_stopping_rounds [int] Activates early stopping. Validation error needs to decrease at least every <early_stopping_rounds> round(s) to continue training. Requires at least one item in evals. If there's more than one, will use the last. Returns the model from the last iteration (not the best one). If early stopping occurs, the model will have three additional fields: bst.best_score, bst.best_iteration and bst.best_ntree_limit. (Use bst.best_ntree_limit to get the correct value if num_parallel_tree and/or num_class appears in the parameters)

self [object] Returns self.

CHAPTER

FOUR

GRADIENT DESCENTS

4.1 Bi-Convexity of the Loss function

TODO

4.2 Gradient Descents

TODO

`utilities.GradientDescents.`**`gradient_descent_theta`**(*theta*, *X*, *U*, *Y*)

Returns the gradient of theta :param theta: a dictionary :param X: hidden variables :param U: input data :param Y: output data :return: grad_theta: dictionary containing gradients of elemnts in theta

CHAPTER

FIVE

DUAL ASCENTS

CHAPTER

SIX

PREDICTING

** section 2 Picard iterations **

class IDL.**IDLModel**(*hidden_features=1*, *alpha=0.1*, *epsilon=0.1*, *random_state=0*, *seed=None*)

demo_param [str, default='demo_param'] A parameter used for demonstation of how to pass and store paramters.

predict(*X*)

A reference implementation of a predicting function.

X [{array-like, sparse matrix}, shape (n_samples, n_features)] The training input samples.

y [ndarray, shape (n_samples,)] Returns an array of ones.

CHAPTER

SEVEN

EXAMPLES

7.1 Classification : MNIST

7.2 Regression : Boston Housing

CHAPTER

EIGHT

CITING

- "Implicit Deep Learning" Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, arXiv:1908.06315, 2019.
- "Bi-convex Implicit Deep Learning" Bertrand Travacca, October 2019

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

u

INDEX

F

G

I

P

U